Applying Evolutionary Algorithms and the No Fit Polygon to the Nesting Problem

Edmund Burke
Department of Computer Science
The University of Nottingham
Nottingham NG7 2RD UK

Graham Kendall
Department of Computer Science
The University of Nottingham
Nottingham NG7 2RD UK

Abstract - *In [6] solutions for the nesting problem are produced using the No Fit Polygon (NFP), simulated annealing (SA) and a new evaluation method. It showed that SA could out perform hill climbing, thus suggesting that evolutionary approaches produce better solutions than a standard search algorithm. In this paper this work is developed. Genetic algorithms (GA) and tabu search (TS) are compared with the results already obtained with SA. The evaluation method is described, along with a description of the NFP. Computational results are given.*

Keywords : Evolutionary, Nesting, No Fit Polygon

1. Introduction

The nesting problem can be described as having to place a number of smaller shapes onto a larger shape (sometimes called a stock sheet). In doing so the smaller pieces must not overlap one another and must stay within the confines of the larger shape. The usual objective in placing the smaller pieces is to minimise the waste of the larger shape. Only two dimensions (height and width) of the shapes are considered. It is also allowable to consider the larger piece to be of infinite height so that only the width of the placements need be checked.

In this paper a number of assumptions are made. The height of the bin (the larger piece) is infinite, although the aim of the evaluation function is to minimise this height. This is common in industry where shapes are cut from rolls of material. Only convex polygons are considered. Only one bin is used. Only guillotine cuts are allowed (that is, a cut must be made from one edge to the other). In later research, once we have shown the method to be effective, we will relax these constraints.

[2] is one of the first references to discuss the nesting problem. In the paper the No Fit Polygon (NFP) is introduced (described in section 2). [3] also used the NFP to calculate the minimum enclosing rectangle for two irregular shapes before placing the rectangle onto astock sheet.

[1] used a heuristic search method to place irregular pieces onto stock sheets. The search is based on the A* algorithm but some restrictions are introduced due to the size of the search space. For example the heuristic function is not admissible. Their evaluation function, f(n), is given by g(n) + h(n) where g(n) is the cost of the solution so far, which is a measure of the waste that cannot be used in later placements. h(n) is a heuristic measure that estimates the amount of waste in the optimal solution should the current piece be included in the placement. If it were possible to find suitable values for h(n) then an optimal solution could be found. However, this is not possible so only “good” solutions can be found. In addition, other restrictions are placed on the search so that the computational times are acceptable. For example, the size of the search tree is limited and the number of successor states is also limited. The Albano and Sapuppo approach makes use of the no fit polygon when deciding where pieces should be placed.

More recently (Oliveira, 1998) also used the no fit polygon. Pieces are placed onto the stock sheet one at a time. The location of the next piece is calculated using the no fit polygon. Once the best placement has been found the piece is added to the partial solution and the next piece is placed. Oliveira experimented with three evaluation functions when considering the placement of the next piece. These being, the minimization of the rectangular enclosure of the two pieces, the minimization of the length of the rectangular enclosure and maximization of the overlap between the rectangular enclosure of the two pieces.

Using the no fit polygon it is not necessary to check for overlap. However, some researchers allow overlapping pieces, at least

during the search phase, although a final, feasible solution must be free of overlaps.
[20] allows pieces to overlap and uses simulated annealing to move a single piece to another position. The objective function considers the length of the layout and combines this with a penalty term that penalises overlap.
[4] also used a meta-heuristic algorithm (tabu search). The approach is similar to [20] except that only non-overlapping pieces are moved.

This paper presents a method to produce solutions to the nesting problem. We employ evolutionary algorithms (genetic algorithms and tabu search) and compare our results with simulated annealing [6].

2. No Fit Polygon

The No Fit Polygon (NFP) determines all arrangements that two arbitrary polygons may take such that the polygons touch but do not overlap. If we can find the NFP for two given polygons then we know we cannot move the polygons closer together in order to obtain a tighter packing. In order to find the placements we proceed as follows (see figure 1).

- One of the polygons (P_1) remains stationery.
- The other polygon, P_2, obits around P_1, staying in contact with it but never intersecting it.
- Both polygons retain their original orientation. That is, they never rotate.
- As P_2 moves around P_1 one of its vertices (the reference point – shown as a filled circle) traces a line (this becomes the NFP).

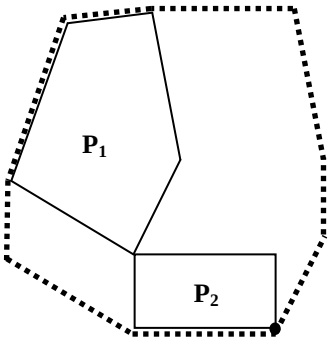

Figure 1 – No Fit Polygon

Figure 1 shows the starting (and finishing) positions of P_1 and P_2. The NFP is shown as a dashed line. It is slightly enlarged so that it is visible. In fact, some of the edges will be identical to some of the edges of P_1 and P_2.

Once we have calculated the NFP for a given pair of polygons we can place the reference point of P_2 anywhere on an edge or vertex of the NFP in the knowledge that P_2 will touch but not intersect P_1.

In order to implement an NFP algorithm it is not necessary to emulate one polygon orbiting another. [8] and [9] present an algorithm that works on the assumption that (for convex polygons only) the NFP has its number of edges equal to the number of edges of P_1 and P_2. In addition, the edges of the NFP are simply copies of the edges of P_1 and P_2, suitably ordered. To build the NFP it is a matter of taking the edges of P_1 and P_2, sorting them and building the NFP using the ordered edges.

3. Evaluation

3.1. Placing the polygons

In order to fill the bin we proceed as follows. The first polygon to be placed is chosen and becomes the stationery polygon (P_1 in the example above). The next polygon (P_2 from the above example) becomes the orbiting polygon. Using these two polygons the NFP is constructed.

The reference point of P_2 is placed on each vertex of the NFP and for each position the convex hull for the two polygons is calculated. Once all placements have been considered the convex hull that has the minimum area is returned as the best packing of the two polygons. This larger polygon now becomes the stationery polygon and the next polygon is used as the orbiting polygon. This process is repeated until all polygons have been processed.

As each large polygon (i.e. output from the convex hull operation) is created, its width is checked. If this exceeds the width of the bin, then a new row within the bin is started. In this case the polygon which forced the width of the bin to be exceeded becomes the stationery polygon. That is, the large polygon built thus far forms one row and the next row is constructed using a single polygon as a starting point.

There are a number of points that need to be considered when using the above evaluation method. For reasons of space, a complete discussion of these points cannot be provided but they are fully described in [5]. However the main points are summarised below

- A number of computational geometry algorithms have been implemented in order

to effectively manipulate polygons. [5] details these algorithms.

- Executing computational geometry algorithms is computationally expensive. This makes the evaluation function the slowest part of the algorithm. In view of this a cache has been implemented which retains *n* (user defined) previous evaluations. If a solution (partial or complete) is in the cache then the results are retrieved from the cache rather than executing the evaluation function. [5] explains how the cache has been implemented and shows how it significantly reduces the execution time of the algorithm.
- To further improve the algorithm the concept of a polygon type has been introduced. This allows polygons which are identical to be classified together so that the cache can work at a higher level of abstraction when deciding if a partial or complete solution is in the cache. [5] provides further details.
- Once the NFP has been calculated there may be a number of optimal placements for the two polygons being evaluated (that is, the convex hulls have equal area). In this case, the convex hull to use is selected at random. However, all the optimal solutions are stored in the cache so that if the solution (partial or complete) is seen again, one of the other convex hulls could be selected. [5] provides details.
- If *n* polygons have been evaluated the solutions in the cache for the n^{th} polygon will be influenced by earlier evaluations. However, even though the convex hulls with the lowest area has been chosen earlier in the evaluation, this may not be the best choice once other polygons are added to the solution. Therefore, it may be beneficial to reevaluate a complete solution even if it is stored in the cache. In order to accommodate this a reevaluation parameter was introduced which, with some probability, forces a solution to be re-evaluated even if it is in the cache. [5] contains further details and shows that this parameter can be set to a low value.

3.2. Evaluating the placement

Our cost function is based on that used in [11]. It can be stated as follows

$$(\sum_{1}^{n}((1-(UsedRowArea/TotalRowArea))^2)*k)/n$$

Where *UsedRowArea* is the total area of the polygons placed in the bin for that row
TotalRowArea is the total area of the stock sheet occupied by that row
k if a factor simply to scale the result. We used 100 but 1 could be used
n is the number of rows in the bin

In words, we are trying to minimise the area used by each row.

4. Search Algorithms

4.1 Tabu Search

Tabu Search (TS) is a search algorithm that maintains a memory of where it has previously visited in the search space. It does not allow the search to return to a state in its memory (tabu list) for a certain number of iterations. This forces the search to explore areas of the search space that have not been explored before. In making a move to a new state TS considers its neighbourhood states (or a subset of them) and moves to the best one, taking into account the tabu list. The move is made even if the new state is worse than the current state. This allows it to escape local minima. [13] and [14] contain a description of TS and a more up to date treatment can be found in [15].

In our TS implementation the neighbourhood function is a swap of two random polygons to give a new permutation. Experiments were conducted with various neighbourhood sizes and a value of twenty gave good results.

The data stored in the tabu list can have an effect on the search. We tested two different methods. Initially just the polygon identifier was stored, which effectively stopped that polygon being moved again until it was removed from the tabu list. This did not perform very well as it restricts the search too much. This is borne out by the fact that the tabu list can, at maximum, store *number_of_polygons* entries, which constrains the search too much.

Another option is to hold the complete solution in the tabu list. It was found that this

increased the run time of the algorithm (due to checking if a move is tabu) but we could achieve equally good results by making moves tabu based on the first *n* polygons in a solution. We achieved the results presented below with *n*=3. That is, if the first three polygons are the same then a move to that state is tabu.

4.2 Genetic Algorithms

Genetic algorithms are search algorithms based on the principles of natural selection and survival of the fittest. GA's were first introduced by John Holland [17]. [16], [10], [12], [18] and [19] provide good introductions.

A GA attempts to evolve a solution using a population of potential solutions (or individuals). New individuals are created by promising genetic material from one individual being passed to another by a process of breeding. Each solution has a fitness value associated with it. Our fitness value is derived from the cost function presented in section 3.2. The fitness of a solution determines how likely it is to be chosen to breed with another solution.

As GA's have their foundations in genetics, terms from this field are used to describe the various features of a GA. It is usual to call a solution a chromosome and the individual parts that make up the chromosome, a gene. For this problem a chromosome is a sequence of polygons and a gene is an individual polygon. Breeding between chromosomes is carried by two operators. Crossover is the most important. It takes two chromosomes (parents) and transfers genetic material from the parents to produce two new chromosomes (children).

We have used the PMX crossover operator [16]. Other operators were tested but it was found that PMX gave the best results. This is not surprising as PMX is designed for ordering problems. As this problem relies on the ordering of the polygons it falls into this category.

The mutation operator works on a single chromosome. It is applied to each child with some (low) probability and aims to stop the population converging to a single solution by adding diversity to the population. We have tested two types of mutation which we call heavy and light mutation. Heavy mutation is applied to a child solution. A gene is picked at random and is swapped with another randomly selected gene, from the same chromosome. This is done as many times as there are genes in the chromosome. Light mutation carries out the same random swap but only does a single exchange. The results from using the two types of mutation are shown below.

5. Testing and Results

5.1. Test Data

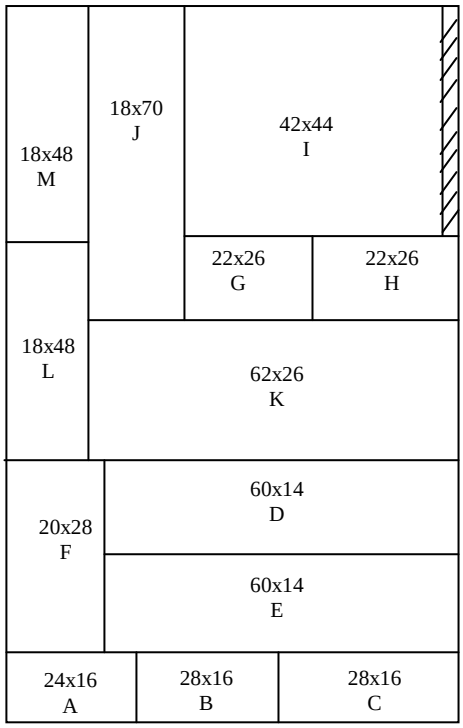

Figure 2 – Test Data 1

Two problems were used in our testing. The packing shown in figure 2 is from [7]. The reason that this data was chosen is because it consists of convex polygons and the optimum is known. The only change made is to multiply the measurements in the original paper by a factor of two. This assists us when displaying the results.

Our algorithms will not be able to find the optimum. The first two rows (A, B, C and D, E, F) can be constructed without problems. However, to find the optimum for the third row the polygons, would need to be presented in the order of G, H, I, J, K, L and M. The optimal solution could be built until the last polygon (M) came to be placed. At this time, due to the convex nature of the large polygon that had been built, the final polygon will not be placed in the position shown. In fact, the final polygon would

be placed on a new row. Under these circumstances the total bin height would be 188 (as opposed to the optimal height of 140).

Therefore, using the permutation ABCDEFGHIJKLM produces a bin height of 188. We accept that we will never achieve a value of 140 but would like to get as close as possible to this figure.

The second problem is taken from the real world. The objective is to cut polycarbonate shapes from larger stock sheets. In reality, the company receiving the orders has to cut many shapes from many stocks sheets whilst minimising the waste. The solution shown (figure 3) is one stock sheet on a given day. However, it is a worthwhile exercise to see how our algorithm packs these shapes. Similar to the first problem, it is constrained by the convex properties of the algorithm. For example, the bottom four shapes cannot be packed in the way they have been as once three of the shapes have been packed, the fourth shape cannot be placed in the position shown. The same is true for the four shapes above (excluding the rectangle).

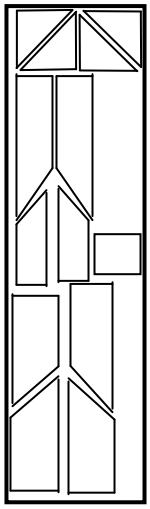

Figure 3 – Test Data 2

The height of the stock sheet is 23940 units. Its width is 6240 units. Due to the convex properties of the algorithm it is not possible to achieve this solution but our main concern is to see if genetic algorithms and tabu search can perform better than the results we have obtained with simulated annealing.

5.2 Results

In this section we compare our simulated annealing results [6] with the results we have achieved using a genetic algorithm and tabu search. All our results are averaged over ten runs. The number of evaluations we performed were equivalent in all cases (SA, TS and GA). This leads to runs of approximately the same time (about 300 seconds on a Cyrix 166 processor) so that the results are compared fairly. In the case of the GA we used a population size of 50 and ran the algorithm for 40 generations. For tabu search we ran the algorithm for 100 iterations, using a neighbourhood size of 20. In the results we show the evaluation value as well as the bin height. Although the bin height is not part of the evaluation, it is an important measure as to the quality of the solution and is therefore worth recording.

The best results we achieved using simualted annealing and the modified Falkenauer function were as follows.

Table 1 – Simulated Annealing Test Results

	Evaluation	**Bin Height**
Test Data 1	397.01	170.80
Test Data 2	1661.77	33052.20

As an initial test for the GA the crossover probability was set to 0.0 and the mutation probability to 1.00. This effectively turns the GA into a random search and the result can be used to check that the GA was actually directing the search.

Two normalization methods were tested. *Evaluation is Fitness* makes the fitness of each chromosome the same as the value returned from the evaluation function. *Linear Normalization* adjusts the fitness of each individual so that the fitness is related to its evaluation but the fitness values are linear throughout the population. Using linear evaluation with roulette wheel selection ensures that the fitter members of the population do not dominate, leading to premature convergence. Better results were achieved with linear normalization.

The two types of mutation described above were tested. Various values for the crossover and mutation probability were tried. It was found that values of 0.8 and 0.1 gave the best results. A dynamic mutation probability was also tested, varying the mutation value between 0.0 and 0.5 during the course of the run. This was found to produce similar results, but no better, than a static value of 0.1.

The results we achieved for the genetic algorithm and tabu search are shown below.

Table 2 – Genetic Algorithm Test Results

	Normalization Method	Crossover Probability	Mutation Probability	Mutation Type	Evaluation	Bin Height
Test Data 1						
	Eval is Fitness	0.8	0.1	Heavy	684.41	192.60
	Eval is Fitness	0.8	0.1	Light	700.10	190.40
	Linear	0.8	0.1	Heavy	610.43	185.20
	Linear	0.8	0.1	Light	516.41	179.00
Test Data 2						
	N/A	0.0	1.0	Heavy	1655.78	32972.40
	Eval is Fitness	0.8	0.1	Heavy	1727.51	32695.50
	Eval is Fitness	0.8	0.1	Light	1607.38	32799.30
	Linear	0.8	0.1	Heavy	1528.06	31353.00
	Linear	0.8	0.1	Light	1377.99	30478.50

Table 3 – Tabu Search Test Results

	Tabu List Size	Evaluation	Bin Height
Test Data 1			
	0	368.03	171.40
	25	342.69	169.00
	50	323.77	165.80
	75	386.68	172.20
	100	386.91	171.00
Test Data 2			
	0	1089.89	28228.50
	25	1101.31	28362.60
	50	1006.75	27566.40
	75	1066.30	28142.40
	100	1200.09	28856.40

From the above results, we can make the following observations.

Of the three methods (SA, TS, GA), GA performed the worst, although a linear normalization function and light mutation produces the best results on both sets of test data.

Tabu search produced better results than simulated annealing for both sets of test data.

The size of the tabu list does not need to be large. Once the list gets too large the quality of the solution starts to deteriorate. We believe this is because the search becomes too constrained. Intuitively, the size of the tabu list should be less than the number of iterations otherwise the search can never visit previously visited states. These results back up that view.

In figure 4 are two of the better solutions we found when using tabu search against test data 1. The nesting on the left has a bin height of 158. The right hand figure has a bin height of 162. Figure 5 is a sample solution, using tabu search on test data 2. This nesting has a bin height of 26430

Figure 4 – Sample Solutions (Test Data 1)

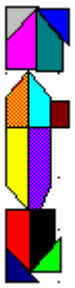

Figure 5 – Sample Solution (Test Data 2)

6. Summary

In this paper a method has been presented to produce solutions to the nesting problem. Our previous work showed that simulated annealing performs better than hill climbing and this work compared two other evolutionary approaches in producing good quality solutions to these problems. We have shown that tabu search produces better solutions than simulated annealing but genetic algorithms perfoms worse than simulated annealing.

We now plan to develop this work further and tackle larger problems as well as non-convex polygons.

7. References

[1] Albano, A., Sappupo, G. 1980. Optimal Allocation of Two-Dimensional Irregular Shapes Using Heuristic Search Methods. IEEE Trans. Syst., Man and Cybernetics, SMC-10, pp 242-248

[2] Art, R.C., 1966. An Approach to the Two-Dimensional Irregular Cutting Stock Problem. Technical Report 36.008, IBM Cambridge Centre.

[3] Adamowicz, M., Albano, A. 1976. Nesting Two-Dimensional Shapes in Rectangular Modules. Computer Aided Design, 8, 27-33

[4] Blazewicz, J., Hawryluk, P., Walkowiak, R. 1993. Using Tabu Search Approach for Solving the Two-Dimensional Irregular Cutting Problem. Tabu Search (eds. Glover, F., Laguna, M., Taillard, E., Werra, D.), Vol. 41 of Annals of Operations Research, Baltzer, J. C. AG.

[5] Burke, E., K., Kendall, G. 1999a. Evaluation of Two Dimensional Bin Packing using the No Fit Polygon. Submitted to CIE26 (Computers and Industrial Engineering), conference to be held December, 1999, Melbourne, Australia

[6] Burke, E., K., Kendall, G. 1999b. Applying Simulated Annealing and the No Fit Polygon to the Nesting Problem. Accepted for WMC (World Manufacturing Congress), conference to be held September, 1999, Durham, UK

[7] Christofides, N., Whitlock, C. 1976. An Algorithm for Two_Dimensional Cutting Problems. Operations Research, 25, 30-44

[8] Cunninghame-Green, R. 1989. Geometry, Shoemaking and the Milk Tray Problem. New Scientist, 12, August 1989, 1677, pp 50-53.

[9] Cunninghame-Green, R., Davis, L.S. 1992. Cut Out Waste! O.R. Insight, Vol 5, iss 3, pp 4-7

[10] Davis, L.D., ed. 1991. Handbook of Genetic Algorithms. Van Nostrand Reinhold

[11] Falkenauer, E. 1998. Genetic Algorithms and Grouping Problems. John Wiley and Sons

[12] Forrest, S. 1993. Genetic Algorithms: Principles of Natural Selection Applied to Computation. Science, vol 261, 872-878

[13] Glover, F. 1989. Tabu Search – Part I. ORSA Journal on Computing, Vol 1, No. 3, pp 190-206

[14] Glover, F. 1990. Tabu Search – Part II. ORSA Journal on Computing, Vol 2, No. 1, pp 4-32

[15] Glover, F., Laguna, M. 1998. Tabu Search. Kluwer Academic Publishers

[16] Goldberg, D.E. 1989. Genetic Algorithms in Search, Optimization, and Machine Learning. Addison-Wesley

[17] Holland, J.H. Adaptation in Natural and Artificial Systems. Ann Arbor: University of Michigan Press, 1975

[18] Michalewicz, Z. 1996. Genetic Algorithms + Data Structures = Evolution Programs (3rd rev. and extended ed.). Springer-Verlag, Berlin

[19] Mitchell, M. 1996. An Introduction to Genetic Algorithms. Massachusetts Institute of Technology

[20] Oliveira, J.F., Ferreira, J.S. 1993. Algorithms for Nesting Problems. In René Vidal, V.V, editor, Applied Simulated Annealing, Lecture Notes in Economics and Mathematical Systems, pp 255-273, Springer-Verlag.

[21] Oliveira, J.F., Gomes, A.M., Ferreira, S. 1998. TOPOS A new constructive algorithm for nesting problems. Accepted for ORSpektrum